Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y-J. Son, eds.

TOWARDS A DEADLINE-BASED SIMULATION EXPERIMENTATION FRAMEWORK USING MICRO-SERVICES AUTO-SCALING APPROACH

Anastasia Anagnostou
[Simon J. E. Taylor]
[Nura Tijjani Abubakar]

Modelling & Simulation Group
Department of Computer Science
Brunel University London
Uxbridge, UB8 3PH, UK

Tamas Kiss
[James DesLauriers]
[Gregoire Gesmier]
[Gabor Terstyanszky]

Centre for Parallel Computing
Faculty of Science and Technology
University of Westminster
London, W1W 6UW, UK

Peter Kacsuk
[Jozsef Kovacs]

Hungarian Academy of Science
Institute for Computer Science and Control (MTA SZTAKI)
Budapest, 1111, HUNGARY

ABSTRACT

There is growing number of research efforts in developing auto-scaling algorithms and tools for cloud resources. Traditional performance metrics such as CPU, memory and bandwidth usage for scaling up or down resources are not sufficient for all applications. For example, modeling and simulation experimentation is usually expected to yield results within a specific timeframe. In order to achieve this, often the quality of experiments is compromised either by restricting the parameter space to be explored or by limiting the number of replications required to give statistical confidence. In this paper, we present early stages of a deadline-based simulation experimentation framework using a micro-services auto-scaling approach. A case study of an agent-based simulation of a population physical activity behavior is used to demonstrate our framework.

1 INTRODUCTION

Recently there is growing number of research efforts in developing auto-scaling algorithms and tools for cloud resources. Usually, traditional auto-scaling performance metrics such as CPU, memory and bandwidth usage are used as an indication for scaling up or down resources. However, these metrics are not sufficient for all applications. For example, modeling and simulation experimentation is usually expected to yield results within a specific timeframe. In order to achieve this the quality of experiments is compromised either by restricting the parameter space to be explored or by limiting the number of replications required to give statistical confidence. In this paper, we present early stages of a deadline-based auto-scaling algorithm implemented in a generic framework for distributed computing resources auto-scaling. The Microservice-based Cloud Application-level Dynamic Orchestrator (MiCADO) framework supports optimal and secure deployment and runtime orchestration of cloud applications.

The motivation for extending MiCADO to enabling deadline-based simulation experimentation is business-driven (Taylor et al. 2018). However, the implementation supports general Bag-of-Tasks (BoT) applications auto-scaling.

A deadline-based experiment (denoted E) can be defined by the 3-tuple:

$$E = (S, t_{deadline}, t_{s\_est}) \text{ where,}$$

- S is the set of simulations s in the experiment, thus $s \in S$
- $t_{deadline}$ is the desired duration of the experiment, and
- t_{s_est} is the runtime estimation of a single simulation s run

We must note here that in the experiment definition the deadline is the desired duration of the experiment. Whether it is a hard constraint, it is not a concern of the experiment definition but of the policy description. Preliminary results show the behavior of the algorithm in restricted and relaxed conditions.

The paper is organized as follows. Section 2 presents related work. Section 3 presents the MiCADO framework and its extension for deadline-based simulation experimentation. In Section 4 preliminary results are presented and finally Section 5 concludes the paper.

2 RELATED WORK

Literature in distributed computing infrastructures show a growing interest in cloud resources orchestration and auto-scaling. Especially, with the growing rate of cloud computing applications (it is estimated that by 2020 83% of enterprise workloads will be in the cloud (Forbes 2018)) and the associated cost in using computing resources as a utility, there is an imperative market need for optimizing the usage of leased resources.

In the realm of deadline-based auto-scaling, a recent survey by Thai et al. (2018) identified that the most popular requirement for BoT applications, including simulation parameter sweep applications, is the deadline constraint with the objective to minimize cost. Cai et al. (2017) performed experiments with deadline-based workflow applications on cloud resources. They have shown that when task execution time is stochastic, the cost of leased resources is increased and often the deadline constraint is violated. They therefore developed a simulator for evaluating deadline-based workflow applications with stochastic task execution time that can be used to evaluate the infrastructure performance. Mao et al. (2010) and Vecchiola et al. (2012) discussed deadline-based cloud auto-scaling in the level of hardware virtualization. They both developed algorithms and implemented them in Azure and Aneka platforms respectively. The literature shows interesting implementations however these are very limited yet.

MiCADO provides functionalities such as virtual machine and container level auto-scaling, programmable scaling logic, standardized application, policy and experimentation descriptions, and resource optimizer. Further, MiCADO provides APIs for various cloud middleware and HPC resources.

In the next section we introduce the MiCADO framework and explain its extension to support deadline-based applications.

3 MICADO TOP-LEVEL ARCHITECTURE

The top-level architecture of the generic MiCADO framework is shown in Figure 1. The architecture is organized in layers based on the concept of microservices as defined in Balalaie et al. (2015) providing decoupling of independent components of monolithic applications. Starting from the top, the *Application layer* contains the actual application code (e.g. simulator) that requires resources on demand. The *Application definition layer* contains application description templates that define the application requirements in terms of infrastructure, security and connectivity. These templates are application agnostic and can be reused by any application with the same requirements. The *Orchestration layer* is padded by the *Coordination interface API* and the *Cloud interface API*. These two APIs decouple the *Orchestration layer*

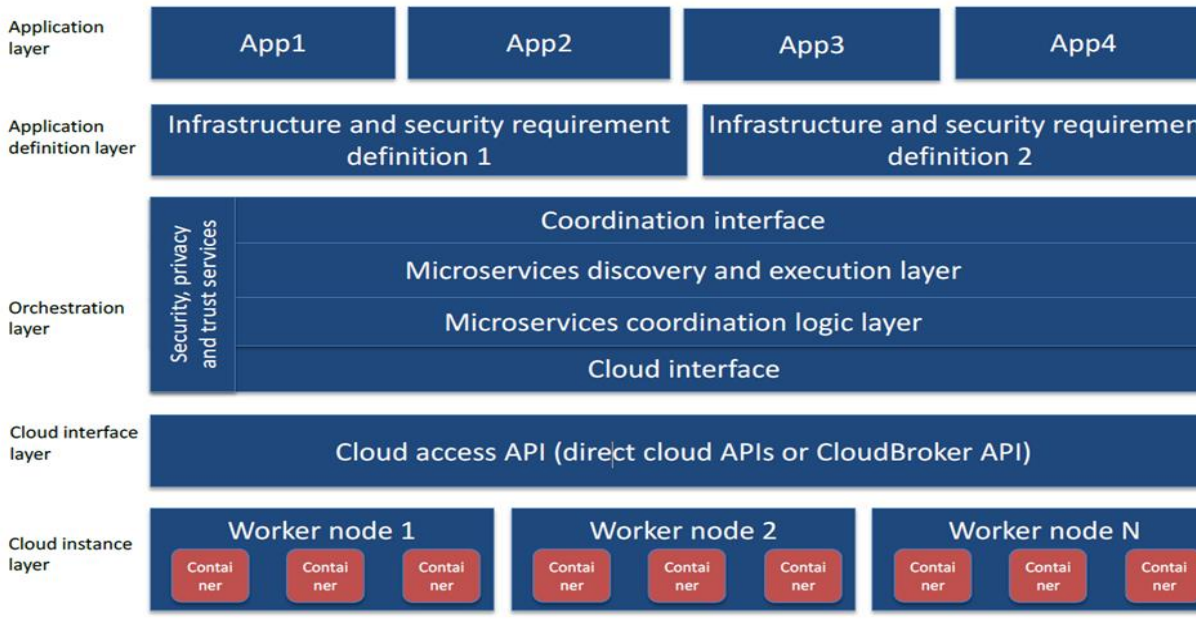

Application layer
App1
App2
App3
App4
Application definition layer
Infrastructure and security requirement definition 1
Infrastructure and security requiremen definition 2
Orchestration layer
Security, privacy and trust services
Coordination interface
Microservices discovery and execution layer
Microservices coordination logic layer
Cloud interface
Cloud interface layer
Cloud access API (direct cloud APIs or CloudBroker API)
Cloud instance layer
Worker node 1
Worker node 2
Worker node N
Contai ner

Figure 1: MiCADO top-level architecture

from the layers above and below, respectively. The *Microservices discovery and execution* sublayer is responsible for starting and stopping microservices execution and keeping reachability information at runtime such as IP address and port number. The *Microservices coordination logic* sublayer is responsible for the operations of running infrastructure such as starting and stopping instances and microservices migration. The *Security, privacy and trust services* vertical layer provides security management across the *Orchestration layer*. The required security services are defined at the *Application definition layer*. The two lower layers, *Cloud interface layer* and *Cloud instance layer*, support access to cloud resources. The first provide interface that supports different cloud middleware APIs for direct access to cloud resources. Further it can interface with Platform as a Service (PaaS) APIs so as additional services, such as billing and account management, can be provided. The second layer contains the actual cloud instances provided by Infrastructure as a Service (IaaS) providers and can be private or public infrastructures. The cloud instances can support both hardware virtualisation and Operating System (OS) virtualisation. In the latter case, many containers can run in the same cloud instance. MiCADO supports both Docker Swarm and Kubernetes container orchestration. More details on MiCADO and its implementation can be found in Kiss et al. (2017).

In the following subsection we describe the general MiCADO design as well as the components that support the deadline-based functionality.

3.1 MiCADO Design

In this section, we provide the high-level design of the core MiCADO framework and its extension to support deadline-based simulation execution. That is, the *Orchestration layer* and the application specific external components that belong to the *Application layer*. At this level of abstraction, we mention the components' functionality rather than the tools used to implement these functionalities. It is worth mentioning however that all components are implemented using open source tools. Figure 2 shows the MiCADO design, which consists of the *MiCADO core* and the deadline-based simulation experimentation application specific components. The functionalities denoted with dotted boxes belong to *MiCADO core* while the ones denoted in clear boxes belong to the simulation application plane. Further, the red outlined components constitute the extension for supporting deadline-based applications.

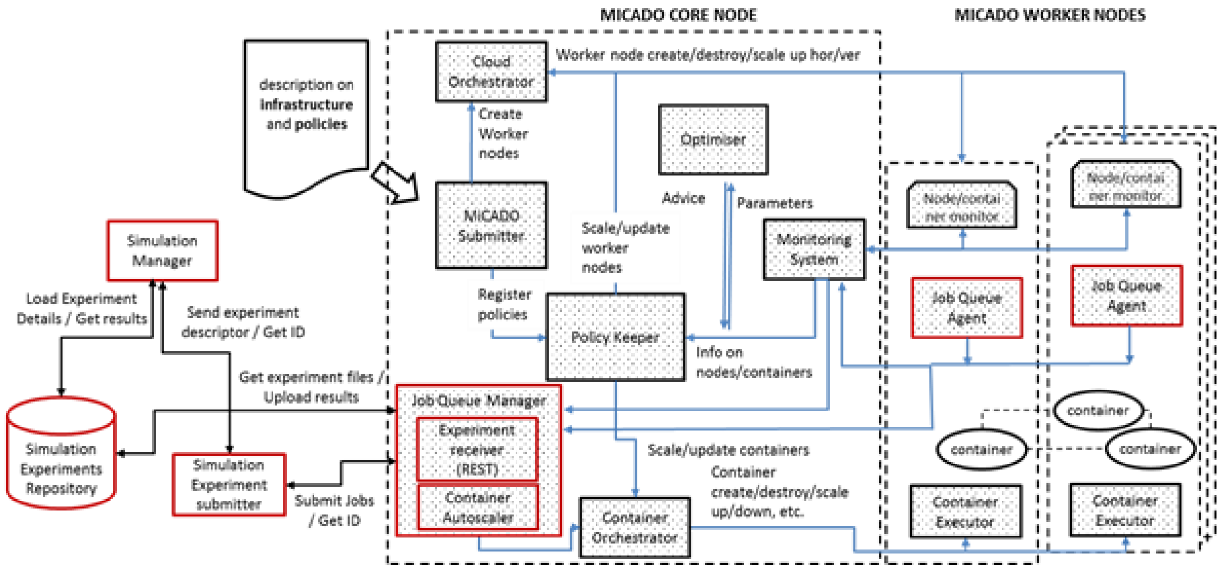


Figure 2: MiCADO high-level design for deadline-based simulation experimentation

3.1.1 MiCADO Core

This is the heart of the *Orchestration layer*. It mainly consists of two logical components: the *MiCADO Master node* and the *MiCADO Worker nodes*. Worker nodes are volatile components where the applications microservices are executing. They are attached to the cluster dynamically and are detached when they are not needed. Decisions on *Worker nodes* allocation are taken in the *Master node*.

The functional components of the *MiCADO Master node* are: *MiCADO submitter* receives the application description (e.g. TOSCA specification), interpreters it and creates the infrastructure. *Cloud orchestrator* allocates and releases virtual machines by communicating with the cloud APIs and starting/stopping *Worker nodes*. *Container orchestrator* allocates applications to containers in the *Worker nodes*, track their execution and destroys them when needed; it is also responsible for scaling at the container level. *Monitoring system* collects information on resources utilization from the *Worker nodes* and creates alerts if appropriate. *Policy keeper* implements the application policies; it makes decisions on starting/stopping cloud resources and on scheduling containers to *Worker nodes*; importantly it keeps *Cloud* and *Container orchestrators* synchronized. *Optimizer* is always running in the background, performs optimization calculations on demand and informs *Policy keeper* on optimized setup of cloud resources and container infrastructure. In the deadline-base implementation, the generic *Policy Keeper* and *MiCADO Submitter* are replaced by the *Job Queue Manager* and its components.

In the *MiCADO Worker nodes* there are three functional components: *Node/container monitor* measures the performance metrics and informs the *Monitoring system* in the *Master node* of these attributes. *Container executor* receives instructions from *Container orchestrator* in the *Master node* to create or destroy containers. Finally, *Container components* realize the applications execution as defined in the container infrastructure description.

3.1.2 MiCADO deadline-based components for simulation experimentation

As mentioned above the deadline-based simulation experimentation application specific components are denoted in the red outlined boxes in Figure 2 and span in the *Orchestration* and *Application layers*. In *MiCADO core* there are two logical components: *Job queue manager* and *Job queue agents*. There are also supporting components in the application layer: *Simulation manager*, *Simulation repository* and *Simulation experiment submitter*. All components are explained below.

Job queue manager communicates with components in both the *Orchestration* and the *Application layers*. Importantly, *Job queue manager* should be able to control the services of the *Container orchestrator*. *Job queue manager* is the component that allocates jobs to containers. It also communicates with the *Monitoring system* to receive runtime metrics and decide whether to scale up/down. Information about the status of each job running in *Worker nodes* is taken from *Job queue agents*. *Job queue agents* are responsible for keeping track of the jobs running in each container in the *Worker nodes*, checking whether there are more jobs to be executed in the *Job queue manager*, fetching new jobs and updating the *Monitoring system*.

Two very important subcomponent of the *Job queue manager* are the *Experiment receiver* and the *Container autoscaler*. *Experiment receiver* is a RESTful web service providing a standard API for experiment submissions. When an experiment is received, an experiment ID is assigned to it. The ID is then sent as a response to the external submitter. *Container autoscaler* is the actual implementation of the deadline-based auto-scaling algorithm. *Container autoscaler* calculates the number of containers needed in order to achieve the experiment deadline and instructs the *Container Orchestrator* to scale up/down. The pseudocode for monitoring the container counter is shown in List 1. The counter checks first whether there are more simulation runs to be executed in the job queue. Then the remaining time to the defined deadline is calculated. Lines 3 and 4 deal with estimating the single simulation run time based on the updated runtime on the cloud resources. Lines 7 to 14 calculate the number of containers needed, restricted by the infrastructure specification (i.e. the minimum and maximum number of containers for the whole experiment), and notify the *Container autoscaler* to start up or shut down containers. The process is repeated in specified time intervals.

List 1: Pseudocode for monitoring the container counter

```
 1   WHILE jobQueue > jobAccomplished
 2     CALCULATE remaining time
 3     IF jobAccomplished NOT ZERO
 4        taskDuration = estimated taskDuration
 5     ELSE
 6        taskDuration = average taskDuration
 7     UPDATE container counter
 8        IF containerCount > JobQueue
 9           containerCount = JobQueue
10        IF containerCount > maxContainer
11           containerCount = maxContainer
12        IF containerNeeded < minContainer
13           containerCount = minContainer
14     SEND to Container Autoscaler
15     WAIT n sec
16   END WHILE
```

The *Application layer* components of the deadline-based simulation experimentation system are: *Simulation manager*, *Simulation experiments repository* and *Simulation experiment submitter*. *Simulation manager* is responsible for uploading experiment details (e.g. simulation model, input data, etc.) to the *Simulation experiments repository*, downloading the results from the repository and creating experiment descriptions that then are communicated with the *Simulation experiment submitter*. The experiment description is a JSON schema providing global experiment data (i.e. experiment deadline, single simulation runtime estimation, URLs for simulation executables, credentials if needed and minimum required resources (i.e. processor, memory)) and specific data (i.e., model, input parameters, output path) for each simulation run. *Simulation experiments repository* keeps all experiments data and files (or URLs to data and files). *Job queue manager* also communicates with the *Simulation experiments repository* for fetching the experiment files and uploading the results. *Simulation experiment submitter* receives the JSON simulation experiment description from the *Simulation manager* and submits the experiment to *Job queue manager* using *Experiment receiver's* REST API. It then gets back the experiment ID and in turn sends it to *Simulation manager*.

More technical details about the Job Queue system can be found in Abu and Kiss (2018).

4 EXPERIMENTATION AND PRELIMINARY RESULTS

In this section, we report preliminary results on evaluating the deadline-based auto-scaling algorithm. We conducted tests with an Agent-Based Simulation developed to study the physical activity behavior of a population. The model takes into account individual characteristics and their effect on physical activity over time. Health conditions risks, such as risk for cardiovascular diseases, diabetes, depression and musculoskeletal incidents, are estimated at a baseline and then are adjusted to reflect the relative risk of the physical activity level on these conditions. of the individual over time. The simulation is built in the REPAST Simphony open source tool (https://github.com/Repast/repast.simphony).

The aim of this paper is to demonstrate our deadline-based simulation experimentation framework that extends MiCADO. We therefore focus on this aspect of the experimentation rather than results related to the physical activity impact on the population health. The runtime of the simulation depends on the simulated population. For our experimentation, we aimed for as short runtime. Therefore we ran the simulation with 15,000 initial population which runs for approximately one and a half minutes.

The cloud resources were provided by CloudSigma (www.cloudsigma.com), an SME cloud provider. Our test model runs for 93 sec on a local Windows PC (i5 3.2GHz CPU, 8GB RAM). For cloud type selection, we first ran the simulation on different instance types aiming to match the execution time of the local PC. The results of the type selection exercise are shown in Table 1. Cloud B is the minimum instance size that gives comparable performance with our local PC, thus we selected this instance for the first performance evaluation. Interestingly, bigger instances show worse performance. We assume that this behavior is due to time-sharing since our resources are not dedicated.

Table 1: Instance type selection

Instance	CPU (GHz)	RAM (GB)	Runtime (sec)
Local PC	3.2	8	93
Cloud A	1	1	440
Cloud B	2	2	96
Cloud C	2	3	92
Cloud D	3	3	175

We then performed experiments for different scenarios combining underestimation and overestimation of single simulation runtime and tight and relaxed deadlines. The number of simulations included in the experiment was 20. We selected a non-scalable hardware virtualization, that is $\min = \max 4\ VMs$ and a scalable OS virtualization of $\min = \max = 6\ containers$. In theory we can run 6 simulations in parallel. However, the selected instance type 2GHz/2GB was too small to run more than one container with our REPAST simulation and its JAVA runtime dependencies. Therefore, the theoretical minimum experiment time is $(96 \sec x\, 20)\ /\ 6 = \ 320\, sec$. Based on the above analysis the experiments 3-tuple elements are shown in Table 2.

Table 2: Experiment scenarios

Scenario	$s \in S$	**t$_{deadline}$ (sec)**	**t$_{s\_est}$ (sec)**
A	20	300	60
B	20	1,200	60
C	20	300	90
D	20	1,200	90
E	20	300	120
F	20	1,200	120

Figure 3 shows the timeline of the numbers of containers indicated as needed by the algorithm and the actual running containers after this suggestion for scenarios B and D (scenarios A and C are not plotted since due to the very short deadline all six containers were running as well as indicated as needed). As we can observe, there is a time elapse between the suggestion and actual scaling. This is to avoid unnecessary starting up and shutting down containers. Another observation is the variation of runtime on the cloud resources we used. For example for scenario D, the average runtime of a single run was 230 sec with a maximum of 543 sec. This behavior caused violation of the deadline.

We should mention here that we do not report on delays of redeployment due to container failures. We plan to include these results in future publications.

5 CONCLUSIONS

In this paper, we presented the implementation of an auto-scalable framework for simulation experimentation. The novelty of our framework is the ability to scale at both the hardware and the OS level. At this early stage, we evaluated the deadline-based scaling algorithm using small-scale experimentation with limited resources. Interesting conclusions have been drawn from the preliminary results that led to our future directions. A very important observation is that the selection of the instance size as per application requirements is critical for optimizing resource sharing.

Our next steps are to evaluate the infrastructure performance with bigger simulations and large-scale experimentation in bigger infrastructures and to automate the generation and execution of experiments. The next version of MiCADO is already implemented and currently is beta testing phase where Ansible playbook is used to orchestrate multi-machine deployment of the parallel infrastructures.

ACKNOWLEDGMENTS

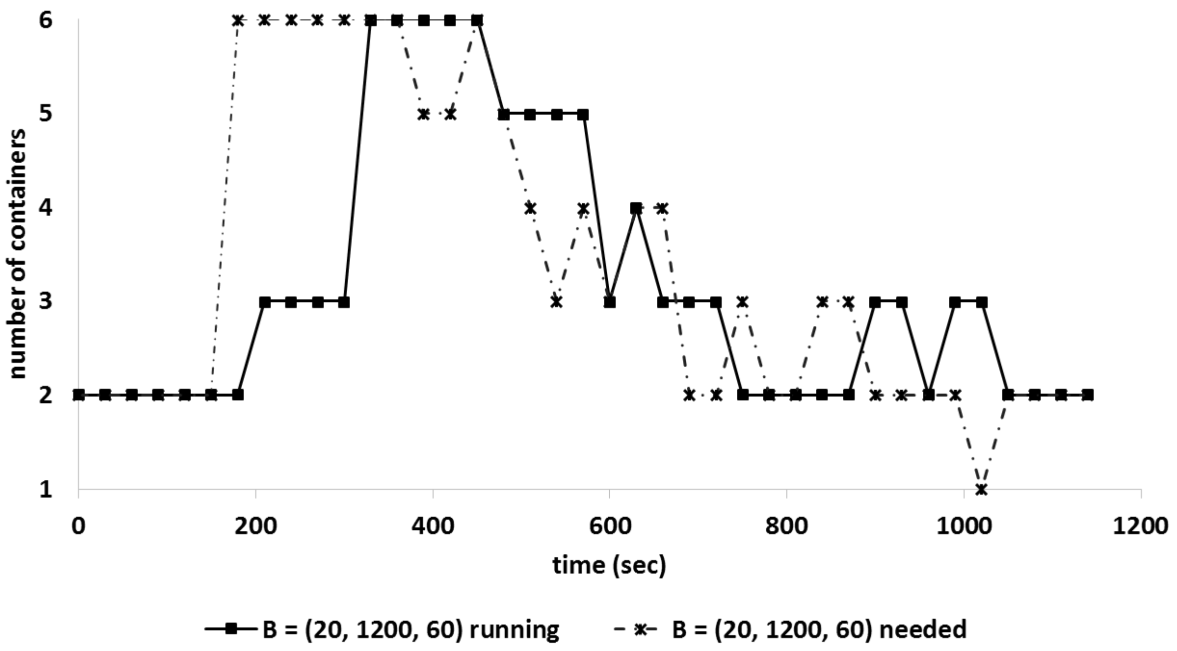
This work was funded by the EU H2020 COLA (Cloud Orchestration at the Level of Application) project (Project no. 731574).

REFERENCES

Taylor, S.J.E., A. Anagnostou, T. Kiss, G. Pattison, S. Kite, J. Kovacs, and P. Kacsuk. 2018. "An Architecture for Autoscaling Cloud-Based Systems for Simulation Experimentation". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 4088-4089. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Forbes. 2018. article by L. Columbus. https://www.forbes.com/sites/louiscolumbus/2018/01/07/83-of-enterprise-workloads-will-be-in-the-cloud-by-2020/#1f69375c6261, 8th April 2019

Thai, L., B. Varghese, and A. Barker. 2018. "A Survey and Taxonomy of Resource Optimisation for Executing Bag-of-Task Applications on Public Clouds". *Future Generation Computer Systems* 82:1-11.



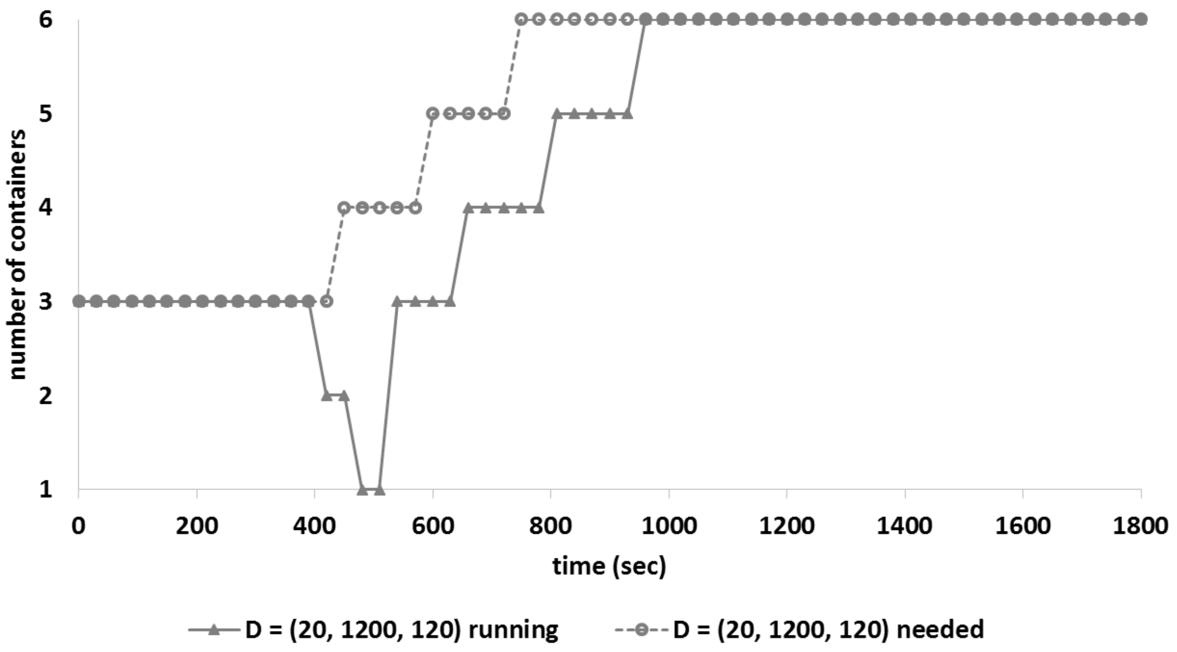


Figure 3: Scenario B & D – Number of containers suggested by algorithm and running in MiCADO

Cai, Z., Q. Li, and X. Li. 2017. “A Toolkit for Simulating Workflows with Cloud Resource Runtime Auto-Scaling and Stochastic Task Execution Times”. *Journal of Grid Computing* 15(2):257-272.

Mao, M., J. Li, and M. Humphrey. 2010. “Cloud Auto-Scaling with Deadline and Budget Constraints”. In *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing*, 41-18. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Vecchiola, C., R.N. Calheiros, D. Karunamoorthy, and R. Buyya. 2012. “Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka”. *Future Generation Computer Systems* 28(1):58-65.

Balalaie, A., A. Heydarnoori, and P. Jamshidi. 2015. "Migrating to Cloud-Native Architectures Using Microservices: An Experience Report". In *Proceedings of the Advances in Service-Oriented and Cloud Computing (ESOCC 2015)* edited by A. Celesti, and P. Leitner. *Communications in Computer and Information Science* 567: 201-215. Springer, Cham.

Kiss, T., P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, and G. Terstyanszky. 2017. "MiCADO—Microservice-based Cloud Application-level Dynamic Orchestrator". *Future Generation Computer Systems* 94:937-946.

Abu Oun, O., and T. Kiss. 2018. "Job-queuing and Auto-scaling in Container-based Cloud Environments". In *Proceedings of the 10th International Workshop on Science Gateways (IWSG 2018)*, 13-15 June 2018.

AUTHOR BIOGRAPHIES

ANASTASIA ANAGNOSTOU is a Lecturer and the Deputy Director of the Modelling and Simulation Research Group in the Department of Computer Science, Brunel University London. She holds a PhD in Hybrid Distributed Simulation, a MSc in Telemedicine and e-Health Systems and a BSc in Electronics Engineering. Her research interests are related to the application of modeling and simulation techniques in the Healthcare and Industry. Her email address is anastasia.anagnostou@brunel.ac.uk and her ORCID is orcid.org/0000-0003-3397-8307.

SIMON J. E. TAYLOR is a Professor and the Director of the Modelling and Simulation Research Group in the Department of Computer Science, Brunel University London. He has a PhD in distributed simulation and has research interests that include modelling & simulation, distributed computing and advancing e-Infrastructures in Africa. He has published over 150 scientific articles, has worked with consortia to attract over £25 Million in research funding, made cost savings in industry of over £3 Million, co-designed numerous industrial distributed computing applications and has recently contributed to African research infrastructures by delivering elements of a roadmap for African National Research and Education Networks across West and Central Africa. He a member of the ACM SIGSIM Steering Committee, founder of the Journal of Simulation and has chaired several major conferences. His email address is simon.taylor@brunel.ac.uk and his ORCID is orcid.org/0000-0001-8252-0189.

NURA TIJJANI ABUBAKAR is a PhD researcher in the Modelling and Simulation Research Group in the Department of Computer Science, Brunel University London. He is investigating how cloud infrastructures can connect and run (geographically) distributed simulations concerning rapid execution of experiments. He has varied professional background in the IT industry which spans over two decades. Before joining Brunel, Nura was a pracademic lecturer in Informatics Institute Kazaure, Nigeria holding a bachelors degree in Computing & Software Engineering from Informatics Academy, Singapore (2008) and MSc in Software Engineering from Oxford Brookes University (2011). He is currently involved in the H2020 Cloud Orchestration at the Level of Application (COLA) project and he is develping Science Gateways for African researchers. His email address is nura.abubakar@brunel.ac.uk.

TAMAS KISS is a Professor in Distributed Computing at the Department of Computer Science and the Director of the University Research Centre for Parallel Computing at the University of Westminster. He holds Master's Degrees in Electrical Engineering, and Computer Science and Mathematics, and Ph.D. in Distributed Computing. His research interests include distributed and parallel computing, cloud, cluster and grid computing. He has attracted over £20 Million research funding and leads national and European research projects related to enterprise applications of cloud computing technologies. He co-authored more than 100 papers in journals, conference proceedings and as chapters of edited books. His email address is t.kiss@westminster.ac.uk.

JAMES DESLAURIERS is a Research Associate at the University of Westminster working on the COLA Cloud Orchestration at the Level of Application project and leading the development activities of the MiCADO orchestration framework at the University of Westminster. His email address is J.Deslauriers@westminster.ac.uk.

GREGOIRE GESMIER is a Research Associate at the University of Westminster. During his Bachelor Degree in computing science, which took place at the University of Besançon in France he did a placement as part of his grade at the University of Westminster. After graduating, in 2014, he joined the ranks of the CPC (Centre for Parallel Computing) of the University where he worked on different project funded under the FP7 body. He worked on developing and supporting application on different technology, such as Cloud computing, Desktop Grid and Cluster. His email address is g.gesmier@westminster.ac.uk.

GABOR TERSTYANSZKY is a Professor in Distributed Computing at the University of Westminster. His research interests include distributed and parallel computing focusing on targeting research issues of Big Data and Cloud Computing. He has been involved in more than 15 research projects as either Principal or Co-Investigator. He also had several research grants at various universities in Germany, Spain, and United Kingdom. He published over 140 papers at conferences and journals. He was member of programming committees of several conferences and workshops. He supervised more than 10 Ph.D. students. He has taught several B.Sc. and M.Sc. modules on distributed computing such as Distributed Computing, Service Oriented Architecture, Web Services. Currently, he is teaching the Client Server Architecture and Advanced Big Data Analytics module. His email address is G.Z.Terstyanszky@westminster.ac.uk.

PETER KACSUK is the Director of the Laboratory of the Parallel and Distributed Systems in the Computer and Automation Research Institute of the Hungarian Academy of Sciences (MTA SZTAKI). He received his MSc and university doctorate degrees from the Technical University of Budapest in 1976 and 1984, respectively. He received the kandidat degree (equivalent to PhD) from the Hungarian Academy in 1989. He habilitated at the University of Vienna in 1997. He received his professor title from the Hungarian President in 1999 and the Doctor of Academy degree (DSc) from the Hungarian Academy of Sciences in 2001. He served as full professor at the University of Miskolc and at the Eötvös Lóránd University of Science Budapest. He has been a part-time full professor at the University of Westminster (London, UK) since 2001. He has published two books, two lecture notes and more than 350 scientific papers on parallel computer architectures, parallel software engineering, Grid and Cloud computing. He is editor-in-chief of the Journal of Grid Computing published by Springer. His email address is Kacsuk.Peter@sztaki.mta.hu.

JOZSEF KOVACS is a Senior Research Fellow at the Laboratory of Parallel and Distributed Systems (LPDS) at the Institute for Computer Science and Control (SZTAKI) of the Hungarian Academy of Sciences (MTA). He got his BSc (1997), MSc (2001) and PhD (2008) in the field of parallel computing. His research topic was parallel debugging and checkpointing, clusters, grids and desktop grid systems, web portals. Recently, he is focusing on cloud and container computing especially on infrastructure orchestration and management. He gave numerous scientific presentations and lectures at conferences, universities and research institutes in many places in Europe and outside. He is reviewer at several scientific journals and holds various positions on conferences. He is author and co-author of more than 60 scientific publications including conference papers, book chapters and journals. His email address is jozsef.kovacs@sztaki.mta.hu.